NAS2-11520

# A TEN YEAR PLAN FOR CONCURRENT PROCESSING RESEARCH

**Prepared by**

**George B. Adams III**
**and**
**Peter J. Denning**

**March 1984**




Research Institute for Advanced Computer Science

84.1

# A TEN YEAR PLAN FOR CONCURRENT PROCESSING RESEARCH

Prepared by
George B. Adams III
and
Peter J. Denning

RIACS
**Research Institute for Advanced Computer Science**

March 1984



---

Prepared under Contract No. NAS 2-11530 by
RIACS
for
Ames Research Center
National Aeronautics and Space Administration

---

# EXECUTIVE SUMMARY

The overall goal of the NASA Computer Science research program, described in NASA Technical Memorandum 85631, "NASA Computer Science Research Program Plan," is building the technical foundation within NASA to exploit advancing computing technology for aerospace applications. That program consists of three core themes: Scientific and Engineering Information Management, Highly Reliable Cost-Effective Computing, and Concurrent Processing. This report focuses on the last of these themes, Concurrent Processing.

Concurrent Processing (CP) is computing with multiple data processing elements cooperating on a single task. Such computers offer significant performance improvements relative to conventional machines. NASA needs very high performance computers for problems in aeronautics and space research such as flow simulation, real-time control, and knowledge-based systems.

RIACS held a workshop at Ames Research Center on September 15-16, 1983 to lay the groundwork for a plan for research and development involvement in CP. This report developed out of the discussions at the workshop and the subsequent review of the participants. The principal conclusions of this report are:

1. The CP research program must strive for understanding of CP systems as complete systems.
2. The research needed to gain this knowledge is inherently multidisciplinary. It follows that the research teams must be multidisciplinary-- with expertise in computer architecture, software, algorithm design, human factors engineering, and the target problem domains.
3. CP research must be based on clear technical principles focused on computational models, mapping of algorithms to architectures, computer architecture, building programming environments, user interfaces, and techniques for system evaluation.
4. Access to simulations facilities and prototype CP machines is crucial to the success of CP research. Systems with many processors are qualitatively different from systems with a few.
5. NASA should make every attempt to cooperate with other CP research programs.

Over the next ten years, NASA will require computers far more powerful than any available today; CP systems are the only way to achieve the extremely high performance needed. Support for CP research is essential for NASA programs and future missions requirements. The intent of this plan is to provide direction for gaining the fundamental knowledge that can substantially advance computer systems for aerospace. Because results of this research can significantly improve NASA's effectiveness, it should be pursued promptly and vigorously.

# TABLE OF CONTENTS

# PREFACE

This report is the outgrowth of a workshop held at NASA Ames Research Center, September 15-16, 1983. The workshop was conducted for Ames Research Center by the Research Institute for Advanced Computer Science (RIACS), an institute of the Universities Space Research Association. The purpose of the workshop was to develop a plan for NASA involvement in concurrent processing research and development. The plan presented in this report is based on input from the workshop and the NASA Computer Science Research Program Plan (NASA Technical Memorandum 85631, March 1983).

The workshop participants included the RIACS staff (George B. Adams III, Robert L. Brown, Roberta A. Cummins, Peter J. Denning, and Eugene Levin) and others listed alphabetically below. Although Adams and Denning took primary responsibility for preparing this report, all these people made significant contributions and should be regarded as coauthors.

James O. Arnold
NASA, Ames Research Center

Bill L. Buzbee
Los Alamos National Laboratory

Donald Calahan
University of Michigan

G. Steve Deiwert
NASA, Ames Research Center

Jack B. Dennis
MIT Lab for Computer Science

James Fischer
NASA, Goddard Space Flight Center

Michael J. Flynn
Stanford University

Robert E. Fulton
NASA, Langley Research Center

Dennis B. Gannon
Purdue University

Paul Kutler
NASA, Ames Research Center

Ronald L. Larsen
NASA, Headquarters

George A. Michael
Lawrence Livermore National Lab

Robert S. Rogallo
NASA, Ames Research Center

Charles L. Seitz
California Institute of Technology

Marcelline C. Smith
NASA, Ames Research Center

Lawrence Snyder
University of Washington

Kenneth G. Stevens, Jr.
NASA, Ames Research Center

Ken Wallgren
NASA, Headquarters

Robert G. Voigt
ICASE

# 1. INTRODUCTION

This report is the outgrowth of a workshop held at NASA Ames Research Center on September 15-16, 1983 by the Research Institute for Advanced Computer Science (RIACS). The workshop was the first step to develop a plan for concurrent processing (CP) research and development. The resulting document (this report) has been delivered to NASA, who will use it as input for their planning processes in concurrent processing.

This report extends and refines the portion of the NASA Computer Science Research Program Plan (NASA Technical Memorandum 85631) dealing with CP. That plan designated research in CP, along with research in Highly Reliable Cost Effective Computing and research in Scientific and Engineering Information Management, as critical to NASA goals. The computer science research program emphasizes research in theoretical and experimental issues critical and unique in aerospace applications. Its major goal is fundamental knowledge that can substantially advance computing systems for aerospace and improve the effectiveness of NASA. In this report, the term "concurrent processing" refers to ***CP systems***, which include user interface, software parts libraries, programming environments, languages, hardware, storage, input, output, and networks. Here "architecture" refers to the structure of the hardware-software system, not just the hardware alone.

## 1.1 Motivation for NASA Support of CP Research

Modern high performance computers can deliver up to roughly one hundred million floating point operations per second (100 MFLOPS), with speeds averaging more nearly 10 MFLOPS. Many of NASA's current activities in science and engineering already demand more power than can be delivered by these computers. The demand for increased computational power is growing rapidly and will continue to grow for the long term future.

This computational need cannot be dismissed as the result of inadequate algorithm design. NASA conducts extensive research and development of algorithms in many disciplines. This is because no closed-form solutions are known for the equations describing many real problems in science and engineering. Computers are being used to extend the reach of mathematics into real applications by making numerical solutions feasible.

Over the next ten years, a computing performance increase by a factor near one thousand will be needed to sustain normal progress in most disciplines; some areas, such as multidisciplinary structural analysis and fluid dynamics, require factors nearer to one million. These requirements cannot be met without computational engines capable of sustaining well beyond one billion floating point operations per second (1 GFLOPS). They also cannot be met without significant

advances in software management technology, data management systems, tools for efficient and correct programming, and mappings from programs to hardware.

Uniprocessor computers are today approaching the fundamental physical limits to their ultimate performance. While circuit switching speeds and packing density will continue to improve, rapid gains are becoming a thing of the past. It is unlikely that uniprocessor technology will provide, by the year 2000, subnanosecond arithmetic cycles (e.g., addition); speeds beyond 1 GFLOPS cannot be expected for single-processor machines. The only way to achieve greater processing speeds is with computer systems employing multiple processing elements. We are not talking about tens of processors acting in concert, but hundreds and even thousands.

A second motivation for NASA's interest in CP is the need for highly compact, reliable computers built in VLSI technology. Not only does VLSI technology make CP systems feasible, concurrent processing is a way to fully exploit the potential of VLSI technology.

There are signs that machine organizations compatible with VLSI implementation, today still in prototypes, will begin to be commercially available by 1986. The initial versions of these machines will be "attached matrix processors" augmenting a uniprocessor; but it is reasonable to expect such designs to mature into stand-alone systems by 1990. These designs will be "linearly expandable," which means their processing power can be scaled upward in equal-cost increments of hardware. One company is talking about pricing its matrix processor at $1000 per MFLOPS.

So the prospects for good CP hardware by 1993 are bright. The real threat to progress toward effective CP systems is the software problem. High performance computer systems of the 1990s will gain their power partly from CP hardware and partly from their ability to aid programming and to bring many software components simultaneously to bear on the solution of a single problem.

The ability to aid the design of concurrent algorithms will require significant advances in visual programming; the goal is to make the construction of a concurrent computation that uses thousands of processing elements no more difficult than the construction of a sequential program of a few hundred lines. Today, for example, the programmer of a solution to a flow problem around a complicated airfoil must manually design the grid and a mapping from it to the 3D computational space. This process can take months to arrive at a correct solution. Concurrent programming systems should be capable of automatic grid generation and mapping within a few minutes.

NASA has many applications which call for existing software systems, each requiring high performance computation, to be redesigned and integrated. For example, there are several existing subsystems that contribute to the design of an aircraft: wing design and flight surface optimization, control system design, overall configuration optimization, structural analysis/design, wind tunnel data analysis, and flow simulation. Each subsystem's computational requirement

alone is sufficient to justify CP hardware; the prospects for joining these subsystems into a single automated aircraft design system is constrained more by software technology than by hardware technology. The ability to integrate different software systems into larger systems will require significant advances in data management, version control, software parts management, and intelligent interfaces.

It is the need for all these software and hardware advances that gives rise to the consensus that CP research is a ***systems*** issue extending well beyond the hardware.

A systems approach to concurrent processing must encompass the problem domains in which the computation will be applied. Each domain must be analyzed for such factors as heterogeneity of the separate processing elements, the degree of potential parallelism, the computational models most suited to the type of parallelism, the need for floating point computation, the need for real-time processing, the need for decision-based processing, the need for symbolic processing, the need for database access, and the I/O bandwidth. A CP system well suited for one application may not be so for another. NASA has a wide range of different applications from numeric to symbolic, each one of which will require careful study.

The importance of a systems approach, exemplified by multidisciplinary research teams, has been expressed several times recently at other CP workshops. The same sentiment was expressed at the RIACS workshop. The participants noted, moreover, that this type of approach is a hallmark of NASA research.

NASA can give as much as it gains by supporting CP research. NASA's applications span a very wide range with respect to types and needs for high performance computation; some applications are virtually unique to NASA. NASA's presence in CP research will bring knowledge and experience not otherwise represented. It will make NASA's extensive expertise in algorithms in the various domains accessible to the larger community.

To summarize: NASA has a wide range of projects, critical to its missions, whose computational requirements exceed anything possible with uniprocessor systems. Concurrent processing systems, compatible with the new VLSI technology and employing new classes of highly parallel algorithms, are, therefore, essential to NASA's success in its missions. The research needed must employ a systems approach that simultaneously pools knowledge from the hardware, software, and applications disciplines; the software problem and the understanding of the applications pose greater impediments to progress than the hardware architecture.

## 1.2 Process of Developing This Report

The first draft of this report was prepared from detailed notes taken at the workshop by RIACS staff. The draft was circulated in November 1983 among all workshop participants, all of whom provided comments, many detailed. A second draft was then prepared and reviewed in February 1984 by the subset of participants requesting this step. The final version of this report was completed in March 1984.

This report appears to represent the consensus of the group. Disagreements are recorded in the few cases where they occurred.

Because the primary purpose of this report is to concentrate on the essence of a research plan, we have deliberately omitted a bibliography.

This document is not a proposal for research to be conducted by RIACS.

## 1.3 Overview of the Following Material

The starting point for the material that follows is a series of scenarios in Section 2 that outline a vision of computing systems for a number of NASA program areas circa 1993. These scenarios reveal three important classes of applications: numeric, symbolic-logical, and real-time.

The research topics that must be encompassed by a CP research program with a systems approach are described in Section 3.

The principles and strategies that will maximize the prospects of a successful CP research program are outlined in Section 4.

# 2. VISIONS OF FUTURE COMPUTING ENVIRONMENTS FOR SCIENCE AND ENGINEERING

## 2.1 Introduction

This section describes computing needs of NASA scientists and engineers in a number of important areas circa 1993. The CP research program must aim for these operational goals.

The material is organized into a series of scenarios, one for each application area. The scenarios are based on research projects at various NASA centers. The areas included are computational fluid dynamics, structural analysis, computational chemistry, image processing, flight simulation, flight control, aircraft design, and space station management computer systems. For each, we have projected the needs of the user communities and the types of computational tools to realize the scenario.

Within each scenario, we distinguish between technological and intrinsic computing needs. Technological needs are those that can be met as soon as the technology base will permit. For instance, the need for magnetic tape as a medium for mass storage and transfer may end with low cost multiple-write optical disks. Intrinsic needs are inherent in the the computations to be performed. For instance, an image processing task will require access to the image data regardless of the type of memory holding the data. As another example, the need to perform $10^{12}$ operations to solve a problem may be intrinsic in the algorithms to solve that problem, whereas the need to obtain a solution within 10 minutes is technological (it can be met as soon as a 3.3 GFLOPS system is available.) We will emphasize the intrinsic needs, which cannot be met simply by better technology.

To permit comparisons of the scenarios, we will explicitly mention the characteristics of operation types, data management, amount and type of data access, computational homogeneity, and interaction with the outside world.

## 2.2 Scenarios of User Activities in Ten Years

### 2.2.1 Computational Fluid Dynamics

NASA's role in the aerospace community has always included fluid dynamics. Digital computer simulation techniques known as computational fluid dynamics (CFD) are an important component of the NASA effort. Computational fluid dynamics has been used to advance understanding of fluid flow and to study performance of flight vehicles. The potential payoff from very high performance computational fluid dynamics facilities includes the ability for engineers to more easily and thoroughly search for aircraft designs optimized for fuel efficiency, maneuverability, and other characteristics. Aircraft designers

will be able to conduct detailed testing of proposed designs, building confidence before committing to construction of models or prototypes.

Workers in this problem domain use the Navier-Stokes equations to model fluid flow. These equations, which appear to be an accurate model of the real world, will continue to be the foundation of this field. To live within current technological limitations on computer speed, researchers have been forced to consider these equations only in simple cases such as inviscid flow. The eventual, but extremely challenging, goal is cost-effective simulations using complete Navier-Stokes equations, including turbulent momentum and heat transport terms. Another goal is to model turbulence on a scale more detailed than that provided by the grid over which the equations are solved.

Future progress in computational fluid dynamics will follow two paths. One deals with fluid flow around more complicated (and realistic) structures. In the past, analyses have been limited to a single component of a proposed aircraft. For example, in the early 1970s, the analysis of an isolated wing strained available computers; today, simulating airflow about an entire aircraft remains beyond the limits of feasible computation. By 1993, researchers and engineers will need to deal with complete vehicle structures.

The second path is the demand for greater accuracy and detail in simulation results. In typical contemporary complete-vehicle simulation, only the terms in the Navier-Stokes formula dealing with inviscid fluid flow are computed. Under certain conditions, however, such as flow at near-transonic speeds, the accuracy of a solution is significantly improved when viscous terms are included. The small-scale flow effects become more important as precision is increased and when aircraft surfaces and configurations must be optimized. To obtain a greater level of detail, flow analysts will require finer mesh spacings, which imply geometric increases in processing capacity.

A hypothetical system fulfilling these needs may have a user interface as follows. A high-resolution (5000 by 5000 pixel) color graphics panel oriented like a drafting table is used to display input and results. The system contains an extensive library of airframe component descriptions that can be retrieved and combined as needed. The three-dimensional constructs can be edited using a light pen or mouse, while working both with standard two-dimensional projections and perspective views. Numeric quantities are entered by pointing to the structure and speaking the digits. This interface should be far less cumbersome than one oriented strictly toward keyboard input.

When the drawing is prepared, the engineer switches to flow simulation mode. Within this mode a collection of solution regions, applying appropriate solution methods to different parts of the structure, is set up. The user environment includes an expert system that advises when the user specifies incompatible solution techniques and automatically computes the interfaces between compatible techniques in adjacent regions. The expert system then proposes a point grid for the problem, based on information about well-formed grids contained in the knowledge base; the user can modify this initial grid if desired. The user then

selects the wind vector, density, temperature, and necessary parameters before starting the simulation. The simulation is run on an underlying high performance computer.

When they are available, the results of the simulation are presented on the color display via animated flow lines overlaid on the image of the aircraft structure. These lines show the positions of features such as shock waves, separated flow, and vortices, which may suggest deficiencies in the design. Optional displays, presented as three-dimensional translucent regions associated with portions of the structure under test, can show important quantities like pressure, density, momentum, or energy. These reports can be saved as input to optimization programs. Using real-time rotation and translation, the engineer can view the structure and overlaid graphics from different angles and distances. If the simulation results contain an error greater than the allowable tolerance, the display flashes in the regions affected. Within a few minutes the information necessary to guide further changes to the proposed design is available.

Computational fluid dynamics problems typically involve large data sets, currently on the order of tens of megawords. In the future these data sets will be much larger; sizes in the hundreds of megawords are projected. NASA estimates that the computational requirements for a realistic flow simulations about a complete flight vehicle are a processing speed of 1 GFLOPS, 32 million 64-bit words of main memory, and 256 million 64-bit words of secondary memory.

Users of aeroflow programs should be able to interact with their executing code at least to the extent of monitoring the progress of a run so that it can be aborted if the partial results are unsatisfactory.

Fluid dynamic calculations are almost always based on floating point numbers. The airflow at a given point depends only on the airflows at neighboring points; the same equation holds throughout a region of space. This type of calculation is well suited for numeric CP systems.

### 2.2.2 Structural Analysis

Structural analysis is a broad discipline that addresses how to calculate stress, vibration, and heat transfer in physical bodies, and the control of these structures. Finite-element models are often used for the calculations. The major applications include nonlinear analysis for large deflections and plasticity (e.g., studies of failure mechanisms), nonlinear transient response (e.g., dynamic failures of flight vehicles), minimum weight structural design, and analyses that include aspects of multiple disciplines (e.g., thermal structural behavior of aeroelasticity). These applications will require from one thousand to one million times greater computational speed than that currently available.

The goal is to model considerably more complex (and realistic) structures at high levels of detail. The largest problem solved to date using finite-element structural analysis was the linear static analysis of an undersea oil storage platform. This problem had over 720,000 degrees of freedom, took about one week

of processing time on a Univac 1110 computer, and generated 850,000 pages of output.

Other representative finite-element structural applications include the analysis of the wing-body intersection for the Boeing 747 and crash analysis of the Piper Navaho light twin engine aircraft. The structural analysis of the Boeing 747 is critical because of the sheer magnitude of the project -- engineers require high confidence in a proposed structural design before they attempt building the prototype vehicle. In the crash analysis of the Piper aircraft, analytical results were compared with data gathered by instruments installed in purposely-crashed vehicles; it helped define adequate structural models for such aircraft and provided an understanding of vehicle crash dynamics.

The major steps in structural finite-element calculations are as follows. A three-dimensional model of the subject structure is input; this is today done by inputting structural member coordinates, connections, material properties, loads, etc., but could be greatly improved with an advanced graphics display system. The matrices defining the properties of each element (node) in the structure are generated. The equations for static stress, vibration, buckling, transient response, and heat transfer are solved. The results of these analyses can then be used as a basis for structural optimization to obtain minimum weight designs.

A hypothetical system to support future structural analyses will use a high resolution graphics screen for output; colors can be used to show changing forces throughout the structure as simulated loads are applied; weak elements can be detected and highlighted; excessive deformation of a structure under stress can be displayed. Hidden line and plane removal is be done in real-time as the user moves the point of view about the structure under investigation. For input, geometric primitives are provided in the user interface environment so that users can construct complex structures easily from substructures whose descriptions are stored in a library.

A baseline problem today in this field is a structure with 10,000 elements and 10,000 equations. Future problems will be much larger. Calculations for vibration and buckling analysis can be particularly lengthy because they compute eigenvectors.

Problems in this domain tend not to be homogeneous, but they often are decomposable into many homogeneous substructures. They require high-precision floating point calculations and large data sets; they require frequent access to their data. Because the substructures can be solved separately, an attractive CP system for static structural analysis might be composed of hundreds or thousands of processors of moderate power. A similar situation occurs for nonlinear transient response where moderate bandwidth communications among neighboring processors is required. For eigenvalue problems, decoupled parallel computations are not so easily utilized and other CP system configurations might be more appropriate.

There is a strong potential for fruitful interaction between computational fluid dynamics and structural analysis. For example, in transonic aerodynamics,

the results of flow analyses could be coupled with structural-response analyses to study wing flutter. Appropriate CP system structures for such multiple discipline analyses must be determined.

### 2.2.3 Computational Chemistry

The goal of research in this domain is reliable calculation of the properties of matter. The subareas of interest to NASA include atmospheric entry physics, combustion, stratospheric chemistry and modeling, climatology, crystallite properties, surface diffusion, chemisorption, catalysis, and materials embrittlement. Computational chemistry is expected to become increasingly important in NASA's research and development efforts.

With current computers, predictions of the properties of gaseous species can be made using Schroedinger's equation. Within current computational limitations, this approach can determine the quantum mechanical behavior of molecules of up to eight atomic nuclei and up to forty electrons. Typical molecular properties computed include potential energy surfaces for ground and excited states, equilibrium geometries, probabilities for transitions between electronic states, and molecular spectra. The information from these computations can be used as input to calculations of other factors such as reaction rates, bulk material properties, and solid-gas interactions. At this level of simulation, computational chemistry can produce results comparable to high quality laboratory measurements.

Quantum calculations can also be performed on systems composed of a gas species interacting with a crystal, e.g., $O_5Ni_{25}$ or $Fe_{54}H$. The technique can be used to model gas species interacting with metal surfaces. The results from these calculations can be compared directly to experiment or can be used to derive effective inter-atomic forces to be used as the basis of atomistic simulations. Larger systems of matter with up to 10,000 atoms can currently be studied using such an approach. The results from this type of simulation are helpful in interpreting experimental results.

To date, results have shown computational techniques to be a valuable addition to the traditional methods of chemistry. Computers and quantum chemistry can provide reliable data for properties of small molecules, nicely complementing laboratory techniques. Computational chemistry also allows investigations that might otherwise be infeasible in the laboratory. Atomistic simulations of materials properties can provide insight into phenomena observed on the macro-scale. Computational chemistry provided useful information aiding design of the heat shield for the Galileo entry probe mission to Jupiter. In a recent space-shuttle experiment, a prediction of this research -- that floridated kapton would resist decomposition from the impact of high energy oxygen radicals -- was confirmed.

The next steps in computational chemistry research are to study the properties of transition metal clusters, repeating components of polymers, and, for ferrous metals, the mechanisms of combustion, catalysis, and hydrogen-accelerated

crack propagation. NASA estimates that such problems will require computer systems with three orders of magnitude greater speed and memory than today's supercomputers.

The calculations needed for these problems require high precision floating point numbers (usually 64 bits). The data sets are large and access patterns not as regular as for computational fluid dynamics problems. The minimum interaction with the user is for termination of bad runs. Dynamic modification of the solution technique for selected regions of a progressing simulation will be a powerful tool.

Because the algorithms used for computational chemistry are less regular than, for example, those of CFD, this domain will require more research into algorithm concurrency. There may also be great potential in reconsidering the algorithms now used for sequential processing.

High resolution graphics figure prominently for displaying results. The user will view three-dimensional molecular skeleton or space-filling models, color-coded to distinguish atomic species, electrostatic field strength, and other characteristics. Zoom, pan, rotation, and cross-section display will allow inspection of molecular surfaces and internal structure.

With a central computational engine capable of 1 GFLOPS, it will be possible to tackle problems such as all-electron quantum mechanical analysis of the kapton monomer, effective core potential analysis of catalysis in materials such as palladium, and 3D atomistic simulation of systems with as many as 50,000 atoms. The programming interface must support a software library and command language allowing simple specifications, in the chemist's terminology, of the desired parameters. Eventually these systems will support molecular engineering.

### 2.2.4 Image Processing

Image gathering systems will continue to figure prominently in NASA missions in the future. The scientific and economic value of past space-gathered imagery guarantees this. NASA missions monitoring weather and earth resources have been outstanding successes, returning data of exceptionally high and immediate value.

In these missions, the flight vehicle has been treated as the collector of data to be analyzed later by computers on the ground. But the sheer volume of data returned places a heavy burden on available computer systems. Future missions will strain computing resources further. In many cases, much of the data is not analyzed until years after its collection. NASA estimates a growth factor of about 100 in the quantity of data collected from orbiting vehicles over the next decade, much of it image-related.

Typical high resolution images contain on the order of 25 million 8-bit picture elements. To reduce the amount of data sent to earth, NASA scientists are studying how to use on-board computers, which will be special-purpose CP systems that extract data only from scenes of interest to human observers. For

example, data from regions of earth obscured by clouds can be summarized or deleted; geometric and radiometric corrections may also be made by on-board computers. There will always be a need to process images in many different and novel ways -- e.g., image enhancements -- that cannot be anticipated during a satellite's design; hence, ground-based image processing systems will remain necessary.

Not only is the volume of data so high that image processing systems capable of output in real time are taxed to their limit, but qualified human analysts are in short supply. Therefore, NASA is also interested in coupling CP image analyzers with expert systems to automate image understanding and identify objects within scenes. A significant amount of progress has already been made with unsupervised image feature extraction to determine ground cover type.

CP systems hold great promise for achieving the processing speeds required for image analyses. A significant amount of research has already been completed in parallel algorithms for image processing. Image computations typically require substantial amounts of integer arithmetic as well as floating point, regular access to very large data sets, and homogeneous processing.

### 2.2.5 Flight Simulation

Simulators are used for flight crew training. They reduce the risk to man and machine of flying aircraft with inexperienced crews. They are also significantly cheaper to operate than the vehicles they simulate. The goal of simulation is to provide the most realistic experience possible to the pilot. Simulators must support mission profiles such as low level penetration, air-to-air combat, mid-air refueling, and landing approaches. Future simulators will require the real-time generation of video displays coupled with real-time response to measurements of the pilot's actions.

Today's flight simulator imagery lacks detail and realism; the techniques include moving a camera over simulated terrain and using a graphics system to generate stick images of terrain. The moving-camera technique suffers from optical distortion and limited viewing fields, both of which seriously limit NASA's important helicopter simulations.

Realistic wide screen images generated by special-purpose CP hardware will be needed in future flight simulators. This equipment must be connected via high bandwidth networks to the computer systems that handle the real-time calculations of the flight vehicle's state and position. Although these calculations are extensive, they are several orders of magnitude less complex than those required for real-time image generation.

### 2.2.6 On-Board Control

Current high-performance aircraft rely heavily on computers for flight control. Several current aircraft are dynamically unstable; without stabilizing computers, they could not achieve satisfactory operation. New levels of performance

and efficiency can be achieved with aircraft that would otherwise be both dynamically and statically unstable.

On-board control computers must satisfy many constraints, such as limits on volume, mass, and power consumption. They must operate reliably in harsh environments where mechanical ruggedness, radiation hardening, and tolerance of temperature extremes are all essential. These computers also have some of the most stringent real-time deadlines known.

While reliable computing per se is outside the scope of this document, research in CP is inescapably allied with highly reliable systems: replicated processing elements present new opportunities to achieve high reliability; many CP projects today include tasks to study the reliability of the new architecture.

Future on-board computers, now being called "pilot's associates," will be able to provide navigation service, optimize flight paths, avoid collisions, generate and update flight instrument displays, monitor long-term vehicle performance, establish and maintain required logs, and automatically receive and record information about airport and airspace changes for future recall.

### 2.2.7 Space Station Manager

The space station will be a complex assembly of various systems, perhaps one of the most heterogeneous designed by man. Some of their functions will include scheduling experiments, life support, navigation, communication, control of experiments, control of manufacturing processes, and troubleshooting. High performance expert systems based on CP architectures may help make these operational decisions. Expert systems technology will figure prominently in these systems because of a strong need for intelligent interfaces and automated decision-making.

A space station manager would monitor all on-board subsystems and maintain links to ground systems. It would detect failed or failing subsystems and attempt reconfiguration to keep their functions operating; it would advise human operators on what repairs to undertake. This expert system would have to cope with all conceivable combinations of failures because the margin for error is too small and the window for rescue too long. This system would monitor the performance of the other subsystems so that gradual performance losses that preceding some types of outright failures can be detected. Such a system would help improve mission effectiveness by reducing unplanned downtime and optimizing schedules for using equipment.

### 2.2.8 Automated Aircraft Design

Currently, at each of the Ames and Langley Research Centers, there are several large software complexes used to design aircraft. These include design and optimization of flight surfaces, evaluation of overall vehicle configurations, structural analysis/design, design of control systems, wind tunnel data analysis, and flow simulation. Each software complex is enormous. Each requires very large computational resources in its own right. Each contains routines, some ten

or fifteen years old, that are badly in need of redesign and modernization. Maintenance of this software often requires excessive time from research staff.

NASA recognizes that its approaches to aircraft design can be made much more effective by integrating these separate software systems and is working toward this. Extensive computer system support will be required -- for revision control, software parts management, common interfaces, and large computations spanning several machines. A expert-system level user interface will be required to help select software parts and combine them into the solution of a given problem, to help manage the development and installation of new software modules, and to organize the many large computations that must contribute to an overall aircraft design.

## 2.3 System Requirements

The above scenarios reveal three basic types of computational needs: numeric (N), symbolic-logical (S), and real-time (R). For each problem domain there is a profile setting forth the extent to which each basic type occurs.

Numeric computations are typically floating-point intensive. They are important in evaluating models of physical processes over grids using a homogeneous description of the relation between each grid point and its neighbors. A CP computer of homogeneous structure with fast communication between connected processors is well suited for these computations. The performance of such computers is often crudely measured by the maximum number of floating point operations per second (FLOPS) of which they are capable.

Symbolic-Logical systems expend the bulk of their processing in retrieving and manipulating the elements of large databases. They have a low density of floating-point operations but a high density of branching operations and I/O operations. A computer with high bandwidth access to the database and uniformity of addressing information is well suited for these computations. The performance of such computers is often crudely measured by the maximum number of logical inferences per second (LIPS) of which they are capable.

Real-Time systems typically have many asynchronous inputs of physical quantities that must be processed and acted on within strict time limits. They are constituted of many near-independent processes of specialized functions. Reliability is very important. The communications environment is distributed. Network bandwidth requirements may vary according to the amount of data acted on by a processor. A central concept of these systems is interrupts, which are the signals that important events have occurred; these signals trigger the invocation of interrupt handlers (processes) whose execution must be completed within given deadlines. The performance of such computers is often crudely measured by the maximum number of completed interrupts per second (CIPS) of which they are capable.

Each of these system types has a different set of demands on a computer system. An architecture suitable for one may not be well-suited for another; an operating system suitable for one may be inadequate for another. Each problem domain may have a mixture of elements of the different types -- for example, an on-board flight control system may combine numeric aspects of image analysis with real-time needs; the automatic aircraft design system may combine numeric and symbolic needs.

An example that combines all three types is a "pilot's associate" for high performance aircraft, as described in the DARPA Strategic Computing program report. Such a system may interpret and respond to image inputs in real time. Another example is an expert system for fault diagnosis and repair of image based systems on the space station.

To summarize: Each problem domain has a profile setting forth the extent to which the domain requires computations of each type. A given problem from a domain can be "sized" by determining for it the capacities in FLOPS, LIPS, and CIPS of the supporting computer system that would guarantee the ability to solve the problem within the desired time.

Table 1 describes the systems of the eight scenarios with a profile showing which types of basic computation appear in each domain.

**Table 1:** Classification of scenarios by predominant type of computation.

| Scenario | System Type |
|---|---|
| Computational Fluid Dynamics | N |
| Structural Analysis | N, S |
| Computational Chemistry | N |
| Image Processing | N, S |
| Flight Simulation | N, R |
| On-Board Control | R |
| Space Station | N, S, R |
| Aircraft Design | N, S |

## 2.4 Observations

Six other observations about future CP systems follow from the usage scenarios. First, the need for greater performance pervades all scenarios. While it is not surprising that the scenarios demonstrate needs for high performance computation -- which was the point of the exercise -- it is enlightening that the speeds sought over the next ten years are three to six orders of magnitude higher than possible with today's supercomputers.

Second, most of the scenarios reveal a need for large memory systems. The data sets arising in most of the numeric profiles are large and growing. Large secondary memory will also be important in applications requiring large components of knowledge-based processing.

Third, most of the scenarios show a strong need for advanced visual displays. Such displays will have features including at least windows; 3D rotation, zoom, translation, and pan; powerful graphics editing; hidden line and surface removal; and perspective rendering. These features must operate in real time.

Fourth, useful systems may combine characteristics of the basic computational types -- numeric, symbolic-logical, and real-time.

Fifth, there is a wide range of problem sizes to be handled by CP. Some problems, especially in real-time applications, may require as few as two processors, and yet are legitimate concerns for CP research. But the really challenging problems for CP research require hundreds or thousands of processors; in these areas, conventional models of communicating sequential processes do not apply. For these domains, linear scaling of capacity by adding hardware increments is critical; the translation of a program to a machine must be parameterized by the resources of the machine.

Sixth, progress in most of the scenarios requires advances simultaneously in several disciplines such as hardware technology, software technology, and concurrent algorithms for the problem domains. Advanced data base management approaches for scientific and engineering information are likely to be important as well.

# 3. RESEARCH PROGRAM DESCRIPTION

## 3.1 Introduction

This section describes the nature of the topics that ought to be studied in NASA's CP research program. In describing these topics, we will emphasize the *process* and the *conceptual base* of a systems-oriented research program. Our intent is to set forth the research themes that must be studied together in a coherent program. In other words, we will address principles, not procedural issues.

An important aspect of systems-oriented research is a sharp focus, within a project, on a well-defined problem domain or set of domains. The project team should comprise experts from each discipline that must contribute -- e.g., scientists and algorithms experts from the problem domain, hardware architects, graphics experts, database experts, operating systems experts, and reliability experts. In the context of NASA, most CP research teams will include persons of all these types.

## 3.2 Research Themes

The following subsections describe research themes that will appear in most CP research projects.

### 3.2.1 Models of Computation

A computational model, which is an abstract representation of the framework in which computations proceed, must be the central focus of the project. In our experience, the most successful computer systems projects have been those in which the designers had, from the outset, a clear conceptual picture of how their machine would perform its computations, how the hardware would implement the model, how the software would map into the model, and how typical problems would be expressed as software.

The computational model must provide a precise description of these concepts:

1. **Data.** What are the atomic objects on which computations operate? How are objects assembled into larger structures? May objects be created and destroyed as the computation proceeds, or are they all known at the start of the computation? How are objects named? How are components of objects selected? How are objects updated in both structure and value? May objects be shared among several operators? What rules govern the retention and lifetimes of objects?
2. **Operators.** What are the basic operators? How are more complex operators built up from simple ones? Is an operator's effect limited to its explicit

operands or may it produce side effects? How is an operation initiated -- by a control signal, by the arrival of operands, or by a request demanding a result? How does an operator obtain operands -- by value, by reference, or by some other means?

3. **Modularity.** To what extent does the model support "abstraction," i.e., the building of pieces from smaller pieces? Can complex data structures be built hierarchically? Can complex operators be built hierarchically? How are new structured pieces saved and made available for future reuse?
4. **Communication.** How do the components of the model exchange information? Are there control signals? Movements of data? Both? Is there a shared memory for all operators? For groups of operators? What data paths are permitted (topology)? Can the data paths be altered dynamically (reconfigurability)? Can data paths be separated into disjoint groups (partitionability)?
5. **Repeatability.** Which components of a computation must have functional behavior (their outputs depend uniquely on their inputs)? Which components are serializable? What happens if a component's execution behavior is not determinate?
6. **Dynamic Behavior and Static Description.** What notation is used to specify the above components and rules (language)? How is the dynamic behavior of a computation described (semantics)? What is the correspondence between static descriptions and dynamic behaviors?

Perhaps the most successful computational model of all time is the "sequential-process model," often called the "von Neumann model" because of the exceptionally clear descriptions of it found in von Neumann's papers. The components are:

1. **Data:** Integer or floating point quantities stored in memory locations of fixed size; memory is a linear array of locations.
2. **Operators:** Atomic operators are those named by hardware opcodes; they obtain their operands only from registers, which can be loaded from memory by special operators. All operators are implemented in the processor. Some operators exchange information with the I/O unit. Operators are encoded as instructions stored in memory. A program counter register in the processor contains the address of the next instruction to be moved into the processor and interpreted.
3. **Modularity:** None in the basic model; each program is self-contained.
4. **Communication:** Information can be transferred between the processor and the memory, or between the processor and the I/O unit. The program counter acts as a narrow channel between an instruction and its successor.
5. **Static and Dynamic:** The static description of a program is a sequence of instructions in assembly language. The dynamic description is the sequence

of steps performed as the program counter steps forward. All program executions are determinate.

This basic model underlies all conventional programming languages, which generalize its basic components. For example, these languages define basic and structured types; they have basic operators but view subroutines as complex operators; they are modular; and they evoke processes as a current statement pointer moves forward.

The basic model has been generalized to encompass collections of executing programs (processes) that exchange timing signals by semaphores and exchange data through interprocess ports or through shared memory. This extension is sometimes called the "communicating sequential processes" (CSP) model. It is one possible model for concurrent processing systems.

A different model for concurrent processing is the data flow model, in which operators "fire" as soon as all their operands are available; the firing of an operator makes results available to successor operands. Static descriptions of data flow computations are graphs displaying the directed data paths between operators. Dynamic descriptions take the form of history arrays that display the sequences of values associated with each output of the graph.

Another model is the Configurable Highly Parallel (CHiP) computer. It consists of a regular array of processing elements (PEs) interleaved with programmable switches. The interconnection structure (connecting each PE to its immediate "neighbors") can be altered to suit each phase of a computation and also to ignore faulty PEs. Static descriptions of computations take the form of graphs constructed on an interactive display. Dynamic descriptions take the form of histories of values at the nodes of these graphs.

In fact, a rich variety of models for CP has been proposed and studied over the past years. These models (and examples of language embodying them) include:

Communicating Sequential Processes (e.g., CSP)
Data flow (e.g., VAL or SISAL)
Functional compositions (e.g., FP)
List processing (e.g., LISP)
Stream processing (e.g., APL, LUCID)
Vector processing (e.g., VECTORAL)
Reconfigurable processor arrays (e.g., POKER)
Logic programming (e.g., PROLOG)
Relational database (e.g., SEQUEL, QBE)

Systolic processing and regular mesh arrays are also models, but since a given array has a fixed purpose, no high-level programming languages (or environments) have been proposed for these cases.

It is important to note that the model associated with a project may evolve with the project over time, and that more than one model may be used in a project.

It is important that a diversity of projects be undertaken, encompassing many models of parallel computation.

### 3.2.2 Mapping of Algorithms to Machines

In addition to providing a precise statement of the syntax and semantics of a given scheme for parallel computation, a model is the basis for evaluating the cost of computation. Computations in the sequential process model, for example, are typically evaluated either for space or time when both the program and data are in random-access store. Computations in a parallel model must use different metrics, such as the maximum number of operations per second for a given amount of hardware; these evaluations must take into account the cost of communicating among processing elements, a factor not important in the sequential process model.

Another cost that must be accounted for are the "losses" from mapping a problem to the model and the model to the hardware. An abstract algorithm in a problem domain can be represented as a graph whose nodes denote operators and edges data paths. If the abstract algorithm's data paths cannot be directly represented in the model, they must be simulated using extra model operators and paths; this will cause a loss due to inability to directly "embed" the abstract algorithm into the structure of the model. There may be a similar loss in embedding the model into the machine on which the model's interpreter is implemented. Very little is known about these losses, and there are some very challenging optimization problems to minimize them.

The problem of determining the cost of computation in different CP models poses some challenging theoretical problems, but experimental studies are also necessary. For each combination of [domain, model, machine] we need to answer: How effective is the model, realized as a language running on a given machine, in helping solve problems in the domain? The most practical approach to finding the answer is to program typical problems from the domain in the language and run them on the machine. Such experiments can produce objective evidence of the costs of computation because the programs and machine performance can be directly measured. They can also produce subjective evidence of effectiveness because the scientists participating in the experiment can be asked whether this form of programming is an improvement, whether the model suggested new or better algorithms, etc. To accomplish this on a large scale -- for many domains, models, and machines -- would be a substantial project.

### 3.2.3 Language

A CP language is the syntax for describing computations in a given CP model. The purpose of a CP language is to permit the parallelism occurring in an algorithm to be expressed explicitly. Attempts to detect parallelism in

programs expressed in sequential languages have been successful at detecting parallelism of degree 10 or 20, but do not appear capable of detecting parallelism of high degree (100 or more processors). The sequential process model obscures parallelism.

Moreover, sequential programming notation is likely to be tedious for expressing parallel algorithms. Much careful thought needs to be given to language design for CP. The goal is to permit expressing highly concurrent algorithms with effort near the theoretical minimum.

An example will illustrate what we mean by the "theoretical minimum." Suppose we have a square region of 2D space, 100 units on a side; we want to solve for the transient flow field when the region is filled with a square mesh of unit side, the value at a node is the average of the values at its immediate neighbors, and all nodes are initial zero except the left edge is initially 1. This description specifies a computation over 10,000 processors. Two possible minimal descriptions are:

1. Devise a syntax that allows declaring square regions, with each of which is associated a mesh size, an initial condition, and a node equation. In this syntax, the program for the computation above would take about as much space as the English word description.

2. Use an interactive graphics system that allows the programmer to draw a square and enter the properties (edge size, mesh spacing, node equation, initial conditions) into slots on a menu.

Once either type of description has been given, the compiler would convert it to a collection of programs, one for each of the 10,000 processors; it would load those programs and provide for initiation of the computation.

The central point is that concepts of the CP model must have compact expressions in the language. The language must take maximum advantage of regularity: the programmer should have to specify an element and its associate rule of replication only once. Interactive graphical methods of programming are more attractive than symbolic linear notations because many concurrent algorithms are naturally expressible as pictures.

To increase its power, the language must support parts-oriented (modular) programming. A part is intended to be a reusable component that can be "plugged in" to any portion of a computation that matches its interfaces properly; parts can be composed into larger parts and saved in libraries. In a sequential model, a part is a closed subroutine with a standard input and output. In general, the definition of admissible parts will depend on the CP model. In the example, squares and rectangles would be admissible parts because they can be plugged together to form a program for a larger region; but circles would not.

The ability to prove that a program meets its specifications will be even more important for CP than it has been for sequential processing. The design of a CP language must therefore consider the ease with which correctness proofs can be carried out automatically.

### 3.2.4 Hardware

A CP model can be implemented on any machine by implementing an interpreter using the assembly language of the machine. Whether this is efficient will depend on the extent to which the machine's concepts are similar to the model's concepts. For example, while list processing is straightforward on LISP machines, on general purpose computers it can be quite inefficient.

The study of hardware in a CP project focuses on direct implementation of the interpreter for a CP model. Prototypes are important because they will help reveal features of the model that defy efficient interpretation, which may lead to a refinement of the model. Prototypes can be used to help evaluate design tradeoffs because alternatives can be tried and their performance measured. For example, the CP model might call for an interconnection structure that cannot be implemented cheaply on the available physical interconnection network; this fact could be fed back to the model and language design processes.

### 3.2.5 Programming Environment

Today the term "programming environment" denotes the collection of facilities the user employs to express, compile, link, debug, and analyze the output of his programs.

We believe that, in CP systems, these functions will be generalized as programming becomes more clearly separated from the use of programs. Programming will continue to rely on tools for expressing, compiling, linking, and debugging program parts; but these tools will require development for concurrent programming. Programming will require a clearer focus on the library of program parts and program templates (partial specifications that will become parts when a few parameters are specified). This implies that CP programming environments will require a database system that allows the location of parts-specifications by queries on their functions, that enforces compatibility of interfaces between parts, and that coordinates revisions and new versions among many contributing programmers. It also implies management procedures that seek to provide reliable, robust, well-tested software parts in official libraries.

The task of using programs -- i.e., composing parts to construct the solution of a given problem and invoking their execution on underlying machines -- will move to a high level and inherit much knowledge of the domain of use. Here there is a great opportunity for expert systems to be deployed at the user interface. These systems can contain knowledge of the problem domain, rules of thumb about which parts are best for what problem instances, and subsystems that construct solutions by plugging together parts selected from the parts library.

### 3.2.6 Fault Tolerance and Reliability

Fault tolerance refers to the masking of hardware failures at the level of the computational model -- i.e., all the operations of the model can be guaranteed to work correctly even if the underlying hardware does not. Fault tolerance does not refer to robustness against errors in programs expressed in the model's language; that is in the province of program correctness.

In many of its missions, NASA has special requirements that hardware systems be reliable. So studies of methods to make CP systems fault tolerant are especially important. NASA has considerable experience in reliable systems that can be brought to CP systems projects.

Increased reliability will also appear as a performance improvement if it necessitates fewer checkpointing operations.

Because many CP systems will be based on highly redundant hardware and will employ VLSI designs, it may be possible to provide a high degree of fault tolerance in the initial design at marginal cost.

### 3.2.7 System Evaluation

System evaluation means two things, both important, in the context of concurrent processing. The first is that we need to rethink performance metrics to be sure they give useful information in the new context. Do the usual system metrics, throughput and response time, extend to concurrent computation systems? How does the cost of communicating among processors affect basic metrics? How can the successful tools of queueing network theory be applied to CP systems? For systems capable of being used in a "pipeline mode" (a second computation can be initiated before the first has cleared the system), are any new metrics or modeling techniques needed?

New systems design projects should include a task to develop a performance model of the system that guides the future designs and the experiments on the current design; and evolves as the design itself evolves. This model can employ analytic and simulation components.

Because the quality of interaction between man and machine is important to many NASA missions, human factors experts should be consultants to, if not participants in, many CP systems projects.

The second meaning of system evaluation arises in the design process. Many American computer design projects pay too little attention to experiment, measurement, and tuning while the system is being designed. A few machines have been designed in which the instruction set is optimized for the most probable patterns of the common compilers and the most frequent operating system operations; and conversely, constraints have been placed on compilers so that statically checkable errors do not have to be detected by hardware.

### 3.2.8 Results Display

Workstations with high resolution interactive graphics displays are highly desirable for many NASA applications. Although the cost of these units is decreasing rapidly for given computational power, NASA cannot afford to be simply a customer for whatever the market happens to offer. NASA needs to maintain personnel who have expertise in modern graphics systems; these persons will be required on CP projects to help put together the advanced user interfaces that meet NASA's specifications. NASA experts in human factors, who have considerable experience from past NASA missions, can help design highly useful display systems and may contribute ideas that can feed back to industry.

### 3.2.9 Special Purpose Machines

There was a debate among workshop participants about the importance of special purpose machines. One group believes that VLSI design technology will eventually become so advanced that "silicon compilers" will be available for creating hardware implementations of algorithms. The other group believes that special purpose hardware will be more cost effective only when general purpose hardware cannot provide needed functions efficiently; they cited special addressing modes (such as bit-reversal in the Fast Fourier Transform) and frequently used complex functions (such as manipulation of Boolean matrices in image problems) as examples.

NASA spaceborne applications have extreme constraints on the size, mass, power consumption, and ruggedness of circuits. These constraints may justify special purpose hardware in flight vehicles that could not be justified for hardware on the ground.

### 3.2.10 Hardware Technology Trends

By hardware technology we mean microelectronics and device technology: materials, fabrication, and packaging. Given the intense interest in these subjects in industry and as part of the DARPA VHSIC and Strategic Computing plans, there is little need for NASA to devote its funds to this topic. On the other hand, NASA needs to keep abreast of the field because it may need to select technologies for compact computer systems for space vehicles.

### 3.2.11 Experimental Machines

We believe that advancement of CP research, especially in algorithms for specific problem domains, requires direct experience with new machines. NASA should acquire such new equipment as parallel machines, attached processors, workstations, displays, and expert systems as they become available and test them in experiments. Understanding the strengths and weaknesses of new designs is a key step in the continuing development of any technology. NASA has a user community with skills in many demanding applications areas and

much experience in evaluating new computer hardware; NASA should more aggressively pursue status as a "beta test site" for such new units as pertain to its CP missions.

NASA can also fruitfully apply its applications and experience in high performance computing by working to develop the design for CP systems; fabrication would be done by an outside contractor. NASA can make the resulting prototype available to the larger research community via computer networks. Another benefit to the Agency would be the development of in-house expertise in system design.

# 4. RESEARCH PROGRAM STRATEGY

We have intentionally stressed policies that should govern CP research projects rather than the details of the tasks that might be contained in these projects. These policies embody a strategy that will maximize the prospects of useful results based on solid conceptual foundations. The principles we recommend are:

1. **Systems Approach.** NASA should support only those CP projects that focus on complete systems comprising concurrent hardware, CP programming environments, software parts technology, and high-level user interfaces that incorporate knowledge of the problem domain in which the system is used.

The systems approach is recommended so that NASA can help the R&D community break out of the pattern of taking each new high-performance hardware unit as an immutable object for which languages must be defined and suitable applications discovered.

2. **Multidisciplinary Teams.** To implement the systems approach, research teams should contain members or active consultants from the various subdisciplines that affect the project. This means the teams should typically contain expertise in hardware, software systems, graphics, concurrent algorithms, human factors, reliability, and the applications domain. (Some team members may be experts in more than one of these specialties.)

Multidisciplinary teams are the hallmark of NASA projects and missions in the past; the same approach should be used in the future.

3. **Strong Conceptual Bases.** CP research projects should be based on clear technical principles. We recommend using the computational model as the focus. Some of the most challenging technical problems will be in understanding the properties and limitations of a model, optimizing the mappings from algorithms to model to hardware, ensuring fault tolerance of the model, building programming environments and parts libraries, and constructing knowledgeable user interfaces.

Experience has shown that projects without solid conceptual bases are likely to fall behind schedule, be difficult to manage, produce unreliable systems, and have little impact when completed.

4. **Experimentation.** Experimentation is important at all stages of a project. This includes building and measuring prototypes; deferring final decisions on language, model, and architecture until the team can adequate evaluate the alternatives and tune the design; and determining by experiment how well each problem domain is served by a given combination of model and machine.

As part of the orientation toward experimentation, NASA should make it a point to acquire new CP hardware as it becomes available and systematically evaluate

it for possible NASA use. NASA should pursue status as a "beta test site" as another way of gaining experience with new CP systems. Even if few machines are found to be useful for NASA, the in-house expertise in advanced architecture thereby accumulated will be worth the investment. NASA may wish to direct construction of prototype CP computers to investigate specific research issues not otherwise addressed in available machines. Prototype CP hardware and software should be made available to the community of CP researchers, not just those within NASA.

5. **Cooperation.** NASA should make every attempt possible to cooperate with other CP research programs, not only those in university and industrial laboratories, but also in other agencies such as DARPA, NSF, ONR, and DoE.

# 5. CONCLUSIONS

NASA is uniquely suited to undertake the research program outlined in this report. It is unlikely that a single industry or university research institution would undertake on its own a program of this scope. NASA has the influence and resources to accomplish this program, and the interdisciplinary project management experience to make the program's components work together successfully. This program is ambitious. But it is doable because many of the needed components already exist. The bulk of the program describes a way to combine existing technologies.

The research to be performed in this program will help provide the computer systems necessary to continue the pace of NASA research and development. In the past, when new, powerful tools were introduced into science and engineering, the rate of new accomplishment increased. CP systems have this potential for NASA.

NASA has maintained a high rate of important technological spin-offs throughout its history. This research program promises to be no different. The cooperative nature of many of the specific endeavors of the program will speed the transfer of knowledge to those outside. The use of CP systems in the various applications outlined will directly benefit those applications.

The technological leadership of the United States has been and will continue to be challenged by the nations of the world. Computers and aerospace vehicles are two areas in which the U.S. has enjoyed the social and economic benefits of preeminence. Continued leadership requires aggressive pursuit of advanced research programs, such as this for CP.